**IN THE UNITED STATES PATENT & TRADEMARK OFFICE**
**BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES**

Appl. No.: **10/058,212**

Applicant: **LAMBERT, Robert J.**

Filed: **January 29, 2002**

Title: **METHOD AND APPARATUS FOR PERFORMING FINITE FIELD CALCULATIONS**

Art Unit: **2131**

Examiner: **ABRISHAMKAR, Kaveh**

Docket No.: **67539/00422**

Board of Patent Appeals and Interferences
U.S. Patent & Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

# BRIEF ON APPEAL

## I. INTRODUCTION:

This is an appeal of the Final Office Action of the Examiner dated May 26, 2006. A Notice of Appeal from the Primary Examiner to the Board of Patent Appeals and Interferences was timely filed with the Office on October 26, 2006, along with a request for a two-month extension of time.

## II. REAL PARTY IN INTEREST:

The real party in interest in the present application is Certicom Corp. The assignment from the Applicant to Certicom Corp. was registered with the office on reel/frame 012840/0504 on April 26, 2002.

## III. RELATED APPEALS AND INTERFERENCES:

There are no related appeals or interferences known to the Appellant, Appellant's representative or assignee which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

## IV. STATUS OF CLAIMS:

In this application, claims 1 and 3-11 are pending, and claim 2 has been cancelled without prejudice. Claims 1 and 3-11 have been finally rejected and are part of the pending appeal. Please refer to Appendix A for a complete listing of the claims involved in this appeal.

## V. STATUS OF AMENDMENTS:

In the response filed by the Applicant on March 16, 2006, claim 1 was amended to recite that upon completion of the computation performed on the machine words, a modular reduction is performed to reduce the result to a predetermined number of words. Claim 2 was cancelled, claims 4 and 6 were amended similar to claim 1, and new claims 7-11 were added.

The above-noted amendments were entered by the Examiner, as indicated in the Final Rejection dated May 26, 2006.

A response was filed by the Applicant on August 28, 2006, however, no further amendments were made.

An advisory action was mailed on September 14, 2006 and a Notice of Appeal filed on October 26, 2006. No further amendments were made.

## VI. SUMMARY OF CLAIMED SUBJECT MATTER:

The claimed invention relates, in general terms, to finite field operations (e.g. see Fig. 8) and to methods for operating on elements in a finite field (e.g. 103 and 113, Fig. 2).

It has been recognized by the Applicant that by not reducing fully as was previously

done, but rather to reduce the result of the computation to a predetermined number of words, effort can be saved and randomness can be added to the representation since, e.g., there are then many ways to write the value, when not completely reduced.

In one aspect, a finite field multiplier (e.g. engine 400 having FF multiplication operator 334, Fig. 5) is provided that is operable to multiply two elements of one of a plurality of finite fields (e.g. 103, 113), where the finite fields are partitioned into subsets. The multiplier comprises a plurality of word sized finite field multipliers (e.g. w0 to wn, Fig. 10), where each is suitable for multiplying elements of each finite field in a respective subset of the plurality of finite fields. The multiplier also comprises a finite field reducer (e.g. 450, Fig. 5), which is configured to perform reduction in the finite field. The multiplier also comprises a processor (e.g. 14, Fig. 1). The processor is configured to operate the finite field multiplier particular to the finite field to be operated on, to obtain an intermediate product (e.g. w2 and w3, Fig. 13). The processor is also configured to operate the finite field reducer on the intermediate product to obtain the product of the two elements (e.g. step 1410, Fig 14) reduced to a predetermined number of words.

In another aspect, a method for adding elements of a finite field $F_{2^m}$ is provided (e.g. see Fig. 12), where $m$ is less than a predetermined number $n$. The method comprises storing a first element and a second element in a pair of registers (e.g. steps 1202 and 1204), where each of the registers comprises the predetermined number $n$ of machine words. An accumulator (e.g. w2 of Fig. 11) is established, having the predetermined number $n$ of machine words, and for each of the machine words in the accumulator, the exclusive-or of the corresponding machine words representing the first and second elements is computed (e.g. steps 1206-1216) to obtain a representation of a result of adding the two elements. Upon completion of the computation, a modular reduction is then performed to reduce the result to a predetermined number of words (e.g. see page 12, lines 20-26).

In yet another aspect, a method of performing a finite field operation on at least one element $r$ and a finite field engine for the same are provided, which also comprise reducing the result to a predetermined number of words.

In yet another aspect, a cryptographic system is provided that is configured for performing the methods described above. The cryptographic system comprises a computational apparatus that performs modular reduction subsequent to running a routine for a plurality of

word sized finite fields.

## VII. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL:

Claims 1 and 3-11 are pending in this application and do not stand allowed.

Claims 1 and 3-11 have been finally rejected by the Examiner under 35 U.S.C. 102(e) as being unpatentable over U.S. Patent No. 6,230,179 to Dworkin et al. (hereinafter "Dworkin").

In the office action dated September 16, 2005, the Examiner rejected original claims 1-6 as being anticipated by U.S. Publication No. 2002/0136402 A1 to Vanstone. In the response filed on March 16, 2006, the Appellant amended claims 1, 4 and 6, canceled claim 2, and added new claims 7-11. Claims 1, 4 and 6 were amended to recite and/or emphasize that upon completion of the computation performed on the machine words, a modular reduction is performed to reduce the result to a predetermined number of words.

The above-noted amendments appear to have overcome the rejection in view of the Vanstone reference, however, the Examiner then finally rejected claims 1 and 3-11 as being anticipated by Dworkin. In the final office action, the Examiner stated that: "Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, this action is made final." [emphasis removed]

In the new grounds of rejection, the Examiner contends that Dworkin teaches each and every element recited in claims 1 and 3-11, including: "upon completion of said computation, performing modular reduction to reduce said result to a predetermined number of words." The Examiner cites column 4, lines 29-34 and column 10, lines 43-53 as teaching such a feature.

A response after final rejection was filed on August 28, 2006 arguing that Dworkin does not in fact teach such a feature and cannot anticipate. An advisory action was issued on September 14, 2006 indicating that the arguments were not persuasive. As such, claims 1 and 3-11 stand rejected.

The Appellant maintains their previous arguments regarding Dworkin, in particular that Dworkin does not teach reducing the result after the computation, and does not operate on machine words. The Appellant therefore respectfully traverses the Examiner's rejections.

## VIII. ISSUES:

The issue on appeal in this matter is whether claims 1 and 3-11 are anticipated under 35 U.S.C. 102(e) by Dworkin.

## IX. GROUPING OF CLAIMS:

The Appellant considers the claims to be separately patentable and, as such, respectfully submits that the claims do not stand or fall together.

## X. APPELLANT'S ARGUMENT:

The Examiner has rejected claims 1 and 3-11 under 35 U.S.C 102(e) as being anticipated by Dworkin. The Appellant respectfully traverses the rejections as follows.

### A. Supplemental Discussion Regarding Claimed Subject Matter

The Appellant has provided a detailed summary of the claimed subject matter in section VI above. To supplement the above summary, the Appellant wishes to further discuss one particular feature recited in, e.g. claim 1, where upon completion of the computation, a modular reduction is performed to reduce the result to a predetermined number of words. As discussed in the specification of the present application on page 23, lines 6-10, the predetermined number of words may be thought of as an indication of, e.g., "how many machine words are needed to store finite field elements". This is exemplified in terms of a finite field multiplication in the specification of the present application at page 16, line 24 through page 17, line 20. In particular, at page 16, line 29 through page 17, line 2, it is emphasized that: "...the multiplication operation is composed of wordsized multiplications. Again the finite field multiplication is composed on wordsized non-reducing multiplications, coupled with a specific reduction engine preferably tailored to the specific finite field." [emphasis added] It is clear therefore that reduction does not take place during the multiplications (i.e. full reduction) but rather a specific reduction takes place at the end.

For ease in understanding this concept, and as would be understood in the cryptographic

arts, reducing to a predetermined number of words can be considered as reducing the result to a "word boundary" to accommodate the sizes of various field elements.

As discussed on page 12, lines 17-26 of the specification as originally filed, the reduction is performed after the routine (or computation) so that the finite field elements may be consistently stored in registers of the same word length, which, as discussed above, can save effort in the computation and add randomness.

Moreover, reducing the result overcomes having to use routines that deal with different exact word sizes or using general purpose code built to handle any number of word components, which is typically slower (see page 3, lines 5-23). Performing bit shifts for each bit of the multiplier results in longer processing time and extra processor operations (see page 4, lines 13-15).

The Applicant has recognized that a predetermined number of words can be chosen to accommodate, e.g. a NIST standard polynomial (e.g. see page 23 as discussed above). When considering word sized values, leading zero coefficients may be added to the full word size and, rather than reducing fully, in the claimed invention, two values may, e.g., be multiplied and then reduced, but only to a particular word size. In this way, the effort required for full reduction need not be expended and only a specific reduction to a predetermined number of words is needed.

For example, before the value is used, and perhaps following many calculations in the "not-fully reduced" form, a final reduction can be made at the end, so as to have a unique value as is required, e.g. by standards to employ in elliptic curve cryptography.

To further assist in the understanding of the above concepts, the Appellant refers to Appendix B – Evidence, wherein a detailed example of how the claimed invention operates in comparison to Dworkin is provided. The evidence has been prepared by Robert J. Lambert.

In summary, the Appellant has recognized that by reducing the result of the routine when performing word sized operations, rather than reducing fully as was done prior, the above efficiencies can be achieved and the above drawbacks can be avoided.

**B. What Dworkin Teaches**

Dworkin teaches a finite field multiplier with intrinsic modular reduction. The multiplier operates using, e.g. a pair of operand registers 42 and 44, and an arithmetic logic unit (ALU) 4.

Operation of the ALU is described in column 4, lines 14-51. It should be noted that the Examiner has relied on portions of this passage in rejecting claims 1, 3, 4 and 6.

The above-noted passage illustrates operation of the ALU for performing finite field multiplication. Two elements *a* and *b*, which are bit vectors (e.g. b=b0,...,bn-1) are multiplied to obtain a product C. It can be appreciated that a bit vector is a single series of bits of arbitrary length, which represents an element. When performing the multiplication, partial products of the multiplicand and each of the bits $b_i$ of the multiplier are formed. It can be appreciated by those skilled in the fields of mathematics and cryptography, that in this context, partial products can be considered pre-accumulated elements (each being reduced), which are then combined to form product C.

As such, in Dworkin, reduction is performed at each step and thus full reduction is performed during the entire operation, which is in fact what is avoided in the present application on appeal. In Dworkin, the partial products are reduced by the modulus if the most significant bit of the previous partial product is set (i.e. if it is a 1) – see column 4, lines 32-34. The balance of the above-noted passage describes repeated use of the modulus register and shifting of the bits. The Appellant, as previously argued, believes that it is clear from this passage that Dworkin teaches reducing the partial products, thus performing full reduction, and does not teach reducing the result to a predetermined number of words as claimed by the Appellant.

The Appellant notes that the passage in column 10, lines 43-53, also relied upon by the Examiner, also clearly teaches reducing the partial products and not the result of the routine.

**C. Appellant's Submission regarding Dworkin**

In order to anticipate under 35 U.S.C. 102(e), each and every element recited in the claims must be found in the cited reference. The Appellant respectfully submits that Dworkin fails to recite each and every element claimed, in particular performing modular reduction subsequent to computing the result and performing word sized operations. As such, Dworkin cannot anticipate.

As noted above, the claims of the present appeal require that the result of the computation/routine/intermediate be reduced to a predetermined number of words rather than by performing full reduction (i.e. reducing at each step as was previously done), to ensure consistent word sizes. Dworkin simply has not recognized the benefits of reducing to a predetermined

number of words and is in fact entirely silent as to word sized operations. Moreover, it is believed to have been shown that Dworkin clearly teaches reducing at each partial product (i.e. at each step of the routine), which is in fact what is avoided by reducing the result at the end, as claimed.

The Examiner further argues in the Advisory Action that column 10, lines 45-52 teaches reducing the result. This is clearly a misinterpretation as this passage again teaches reducing each partial product – i.e. full reduction.

The Appellant notes that the previous rejections in view of the Vanstone reference were argued on a similar point, namely that the Vanstone reference also teaches reducing partial products and does not perform word sized operations. As such, the Appellant believes that the Dworkin reference is no more relevant than the previously cited Vanstone reference, which appears to have been successfully argued over.

It is therefore believed to have been shown that Dworkin clearly does not teach every element recited in claims 1, 3, 4 and 6 (and those dependent thereon). As such, Dworkin cannot anticipate and the rejection under 35 U.S.C. 102(e) is believed to be improper.

The Appellant also notes that, as discussed in detail above, the claims under appeal deal with word sized operations, and reduce the result of the computation to a predetermined number of words. The word sized method claimed in the present application under appeal separates the full reduction out so that reduction is only needed to be applied when desired (e.g. to supply unique representations for ECC algorithms). However, Dworkin only mentions operating on bit vectors and even explicitly teaches reducing at each step, thus performing full reduction. As discussed above, the Appellant again respectfully submits that Dworkin is entirely silent regarding word sized operations and clearly teaches only full reduction at each step.

The Appellant believes that the Examiner is incorrect in interpreting the teachings in Dworkin as word sized operations (see Advisory Action). Operating on word size values rather than bit vectors is not arbitrary, and these terms cannot be considered interchangeable as the Examiner seems to believe. It is improper to reject a claim as being anticipated on the basis of an extrapolation of the teachings of the cited reference, even more so where the Examiner has not pointed to any passage in the teachings that would support such an extrapolation. Dworkin simply does not teach word sized operations and the Examiner has failed to specifically support the basis for this interpretation. Each and every claim in the present application deals with word

sized values and, for at least this reason, Dworkin cannot anticipate.

Accordingly, it is believed that Dworkin fails to teach yet another feature recited in the claims under appeal. For this additional reason, Dworkin cannot anticipate.

**D. Summary**

It is therefore submitted that, contrary to the Examiner's assertion, Dworkin clearly does not teach reducing the result of the finite field operation and, moreover does not teach word sized operations. Dworkin thus fails to teach each and every element claimed, and, as such cannot anticipate.

## XI. CONCLUSION:

In view of the foregoing, the Appellant believes that the Examiner has misconstrued the passages relied upon in rejecting the claims under appeal as being anticipated by Dworkin. In particular, Dworkin clearly does not teach modular reduction upon completion of the accumulation, but rather teaches reducing partial products prior to accumulation. Dworkin also does not perform word sized operations as claimed but is only concerned with bit vectors. Therefore, Dworkin clearly does not teach every element claimed and, as such, cannot anticipate.

The Appellant respectfully requests that this honourable Board of Patent Appeals and Interferences reverse the Examiner's decision in this case and indicate the allowability of claims 1 and 3-11 in this application.

Respectfully submitted,

Date: Feb. 19/07

Brett J. Slaney
Agent for Applicant
Registration No. 58,772

BLAKE, CASSELS & GRAYDON LLP
Suite 2800, P.O. Box 25
199 Bay Street, Commerce Court West
Toronto, Ontario M5L 1A9
CANADA
BSL/

Tel: 416.863.2518

**APPENDIX A:**

Listing of the claims involved in the appeal:

1. (previously presented) A method of adding elements of a finite field $F_{2^m}$, where m is less than a predetermined number n, said method comprising the steps of:

a) storing a first and a second element in a pair of registers, each of said pair of registers comprising said predetermined number of machine words;

b) establishing an accumulator having said predetermined number of machine words; and

c) computing for each of said machine words in said accumulator the exclusive-or of the corresponding machine words representing each of said first and second elements to obtain a representation of a result of the addition of said elements, and, upon completion of said computation, performing a modular reduction to reduce said result to a predetermined number of words.

2. (canceled)

3. (original) A finite field multiplier operable to multiply two elements of one of a plurality of finite fields, said finite fields being partitioned into subsets, said multiplier comprising:

a) a plurality of wordsized finite field multipliers, each suitable for multiplying elements of each finite field in a respective subset of said plurality of finite fields;

b) a finite field reducer configured to perform reduction in said one finite field;

c) a processor configured to

i) operate the wordsized finite field multiplier suitable for use with said one finite field to obtain an intermediate product; and

ii) operate said finite field reducer on said intermediate product to obtain the product of the two elements.

4. (previously presented) A method of performing a finite field operation on at least one element r, of a finite field, comprising the steps of:

a) representing each element as a number of machine words;

b) performing a wordsized operation on said representations, said wordsized operation corresponding to said finite field operation;

c) completing said wordsized operation for each word of said representations to obtain a result; and

d) performing a modular reduction of said result to reduce said result to a predetermined number of words.

5. (original) A finite field engine for performing a finite field operation on at least one element of a finite field chosen from a set of finite fields, said set of finite fields being divided into subsets according to their word size, comprising:

a) a finite field operator for each of said subsets;

b) a finite field reducer for each of said finite fields;

c) a processor configured to choose the finite field operator corresponding to the subset containing said chosen finite field and the finite field reducer for said chosen finite field and apply the chosen finite field operator to said element to produce an intermediate result and apply the chosen finite field reducer to said intermediate result to obtain the result of said finite field operation.

6. (previously presented) A cryptographic system comprising:

a) a plurality of elliptic curves, each specifying elliptic curve parameters and a respective finite field;

b) a plurality of finite field settings corresponding to each finite field;

c) a plurality of wordsized finite fields, each having routines, each finite field being assigned to one of said wordsized finite fields;

d) a reduction routine for each finite field;

e) a computational apparatus configured to perform a cryptographic operation by the steps of:

i) selecting one of said elliptic curves; and

ii) performing a cryptographic function using the routines from the wordsized finite field to which the respective finite field corresponding to said selected elliptic curve is assigned; said routines including at least one finite field

operation and, subsequent thereto, a modular reduction to obtain a result of said operation corresponding to a predetermined number of words.

7. (previously presented) A method according to claim 4 wherein said modular reduction is determined by said finite field.

8. (previously presented) A method according to claim 4 wherein said finite field operation is addition.

9. (previously presented) A method according to claim 4 wherein said finite field operation is subtraction.

10. (previously presented) A method according to claim 4 wherein said finite field operation is multiplication.

11. (previously presented) A method according to claim 4 wherein said finite field operation is division.

## APPENDIX B – EVIDENCE APPENDIX

**Example prepared by Robert J. Lambert:**

The following f is the irreducible polynomial used in the NIST standard sect163k1 curve:

f := x^163+x^7+x^6+x^3+1.

The improvement is to not to reduce fully, but rather only to reduce to a word boundary.

When 32-bit words are used, the next word boundary is at 192 bits. We would then reduce with this shifted polynomial:

fw:= sort(expand(x^29*f));

$$fw := x^{192} + x^{36} + x^{35} + x^{32} + x^{29}$$

It can be awkward to write out these polynomial (mod 2) values, so instead they will be presented as binary numbers. We will use the convention in this writing that a higher order coefficient is written to the left of lower order coefficients, that is, written high to low from left to right. This notation is, of course, arbitrary.

Here are the irreducible and the shifted irreducible written in this way:#

convert(polynomialToInteger(f), binary);

```
10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
\
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
\
11001001
```

convert(polynomialToInteger(fw), binary);

```
10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
\
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
\
11001001000000000000000000000000000000
```

When we are considering wordsized values, we will, when convenient, write out the leading 0 coefficients to the full wordsize. For example, the unshifted modulus here is written out to the full 192 bits using leading zeros:

```
print(polynomialToBinary(f, fw));

000000000000000000000000000010000000000000000000000000000000000000000000000000
\
000000000000000000000000000000000000000000000000000000000000000000000000000000
\
000000000000000000000000000011001001
```

Here are several random values, which are only reduced enough to fit in the 192-bit word aligned limit:

```
a := randomPolynomial(fw):
b := randomPolynomial(fw):
c := randomPolynomial(fw):
polynomialToBinary(a,fw);

111010010010001111111111111011101000111000100011110100001001010100000110101100 0
\
011000011011011010111111010100011001011011010010111100111111110111110001011100
\
101011010011100011101001111101001101

polynomialToBinary(b,fw);

000110110011010001000001110000010101000110110110110100011010011000000100011101
\
100001100111100110010111111011011100011110100111100011000100111111011011110010
\
100000101101110010101111011000111000

polynomialToBinary(c,fw);

010101010101110110101000001100100110000101111111101101001100011010010001011011
\
100010110000100110010101000000110011111011100110010100000111010101111111010000 1
\
111101001110010000011011010111100100
```

We can multiply these values, mod 2, without reducing fully, yielding:

```
abw := multiplyPolynomial(a,b):
convert(polynomialToInteger(abw), binary);

100011110000011011000010001101111111011110010011110000111010110000110101000000
\
000011010100001110100110111000001000111001100011100011011111000001010011011111
```

\
100100110111110001111010100010100110010111001111100010111101101011111000000010
\
110011011011010011101011010111001010011110101111010101110001011111000110111111
\
000001000100111110100010101110011101001000000101011111111100100011000

We still reduce, but only to the wordsize that we are employing, and therefore we don't need to

expend the effort of full reduction. There are multiple 192-bit strings which reduce (mod f) to the same value.

The utility of not reducing fully is, in part, to save the effort, and to randomize the representation, since there are now many ways to write the value, when not completely reduced.

Here is the wordsized reduced value as described in the patent application:

abwr := reducePolynomial(abw,fw):
polynomialToBinary(abwr,fw);

011110010000001110110010000111010100110010110001000110100110000010001111101111
\
101001110000011111100011110110110101100110101110110100110101101001001001001000
\
010000000000010101111111100100011000

There are many ways to arrive at this, for example by wordsized reduction interleaved with multiplication, or by full multiplication without reduction followed by wordsized reduction (which is what we have just performed). In general, any method using only wordsized reduction, not the complete reduction of the older is covered.

Before we use the value, and perhaps after many calculations in the not fully reduced form, we can at the end make a final and full reduction, so as to have a unique value as is required by standards to employ in elliptic curve cryptography. For example, we might add another value:

abcwr := addPolynomial(abwr,c):
polynomialToBinary(abcwr,fw);

001011000101111000011010001011110010110111001110101011101010011000011110110100
\
001011000000111001110110110111010010010001100010011100111011000010110111101001
\
101101001110000101100100110011111100

Here is the complete reduction of the multiplication and addition just given (with leading zeros):

```
abcwr := reducePolynomial(abcwr, f):
polynomialToBinary(abcwr,fw);

00000000000000000000000000000111001011011100111010101110101001100001111011010
0
\
00101100000011100111011011011101001001000110001001110011101100001011011110100
1
\
10001100000011010010111001100101000
1
```

This is now contrasted with the Dworkin reference:

The Dworkin reference would have used the fully reduced representation for the values a and b. The fully reduced representations are given by:

```
ar := reducePolynomial(a, f):
br := reducePolynomial(b, f):
cr := reducePolynomial(c, f):
polynomialToBinary(ar,f);

10100011100010001111010000100101010000011010110000110000110110110101111110101
0
\
00110010110110100101111001111111101111100010111000011111111001010100100011100
0
\
0101110

polynomialToBinary(br,f);

00101010001101101101101000110100110000001000111011000011001111001100101111111
01
\
10111000111101001111000110001001111111011011110010100101011111011111100010110
11
\
1000000

polynomialToBinary(cr,f);

01001100001011111111011010011000110100100010110111000101100001001100101010000
00
\
11001111101110011001010000011101010111111101000011000111011101000000001101010
0
\
1010010
```

When multiplying, the partial products are fully reduced at each step in Dworkin. The partial products of the multiplication of Dworkin are printed completely below (prefixed with the index and comma, and counting down, as was done in the outer loop of the Dworkin teachings):

```
ab  := multiplyPolynomialFullyReduced(ar,br,f):
```

162,

00101010001101101101101000110100110000001000111011000011001111001100101111110\
1\
10111000111101001111000110001001111111011011110010100101011111011111100010110\
11\
1000000

161,

01010100011011011011010001101001100000010001110110000110011110011001011111101\
1\
01110001111010011110001100010011111101101111001010010101111101111110001011011\
1\
0000000

160,

10000010111011011011001011100111110000101011010111001111110011111110010000101\
1\
01011011001001110011011111010111000010110100111000110000100010100001101001101\
01\
1000000

159,

00000101110110110110010111001111100001010110101110011111100111111100100001011\
0\
10110110010011100110111110101110000101101001110001100001000101000011010011010\
10\
1001001

158,

00001011101101101100101110011111000010101101011100111111001111111001000010110\
1\
01101100100111001101111010111000010110100111000110000100010100001101001101010\
1\
0010010

157,

00010111011011011001011100111110000101011010111001111110011111110010000101101\
0\
11011001001110011011111010111000010110100111000110000100010100001101001101010\
10

\
0100100

156,

000001001110110111110100010010001110101111010010001111111100001010001001001000
\
000010101000011110001011011010001001001010111111010110111011100010111100001111
\
0001000

155,

001000111110110100110010101001010001011100101010101111001011100111011001101101
\
101011011111110111110011101011000110111100000011111111110110001010100010010001 01
\
1010000

154,

011011011110110010111111011111101110111011011011101110100100111101111000100110
\
111000110000001100111111001110000100011101110110101100011110111011100011010000
\
1100000

153,

110110111101100101111110111111011101110110110111011101001001111011110001001101
\
110001100000011001111110011100001000111011101101011000111101110111000110100001
\
1000000

152,

101101111011001011111101111110111011101101101110111010010011110111100010011011
\
100011000000110011111100111000010001110111011010110001111011101110001101000010
\
1001001

151,

011011110110010111111011111101110111011011011101110100100111101111000100110111

\
0001100000011001111110011100001000111011101101011000111101110111000110100001001
\
1011011

150,

1111010011111101001011011101101000101101001101010110011111001011010000100100111
\
1000100011000111000000100000110110001100000100100101010000010101110001010100101
\
1110110

149,

1110100111111010010110111011010001011010011010101100111110010110100001001001111
\
0001000110001110000001000001101100011000001001001010100000101011100010101001001
\
0100101

148,

1101001111110100101101110110100010110100110101011001111100101101000010010011110
\
0010001100011100000010000011011000110000010010010101000001010111000101010010011
\
0000011

147,

1010011111101001011011101101000101101001101010110011111001011010000100100111001
\
0100011000111000000100000110110001100000100100101010000010101110001010100100111
\
1001111

146,

0110010111100100000000111100101100001001111011000101111111000100011101111000101
\
0011010010000100110100010101000100111010010111000000101110100111101001011111101
\
0010111

145,

11100001111111101101010100011000111001110011111110111100001011010001010111011
\
11010001111111010101001100101011100011111100000101011101101101001011101010000 1
\
1101110

144,

11101001110010110111000000000101000011101111000110111011011001101110000000100 10
\
00011011000011100101011111011110111001001111101111110001100100101000010001100 1
\
1010101

143,

11111001101000000011101000111111011011101011011011011010111110001000010110110 01
\
10001110111010000101111000110100001100101000111010101001110111101111100110100 1
\
0100011

142,

11110011010000000111010001111101101110101101101101101011111000100001011011001 1
\
00011101110100001011110001101000011001010001110101010011101111011111001101001 1
\
0001111

141,

11001100101101100011001011001111101101010011100000010100111110001110011001101 1
\
10000011010101011000100101011001001100010100001111101101100000000000101111111 00
\
0010111

140,

10011001011011000110010110011111011010100111000000101001111100011100110011011 1
\
00000110101010110001001010110010011000101000011111011011000000000001011111110 01
\
1100111

139,

00110010110110001100101100111110110101001110000001010011111000111001100110111
0
0000110101010110001001010110010011000101000011111011011000000000010111111110010
0
0000111

138,

01100101101100011001011001111101101010011100000010100111110001110011001101110
0
0001101010101100010010101100100110001010000111110110110000000000101111111001000
0
0001110

137,

11001011011000110010110011111011010100111000000101001111100011100110011011100
0
0011010101011000100101011001001100010100001111101101100000000001011111110010000
0
0011100

136,

10111100111100001000001111000010011001111000110001011100001000000000011000110
1
1101001001000101110110101010111111010011000001001111101011111001000011110010100
0

0110001

135,

01111001111000010000011110000100110011110001100010111000010000000000110001101
1
1010010010001011101101010101111111010011000001001111101011111001000011110010101
0
0101011

134,

11110011110000100000111100001001100111100011000101110000100000000001100011011
1

0100100100010111011010101011111101001100000100111110101111100100001111001010101
\
1010110

133,

1100110110110010110001000010011111111100111011000010001000111100111110100100 11
\
0010101011011010001001001111011101100011010111101001110100110011100010000011 11
\
0100101

132,

1001101101100101100010000100111111111001110110000100010001111001111101001001 10
\
0101010110110100010010011110111011000110101111010011101001100111000100000111 11
\
0000011

131,

0001110011111110111001010101010110011001100111111001001011110011110010001011 0001
\
0001001110011100011000100101010001110110000000110011111000110101110100011001 00
\
0001111

130,

0011100111111011100101010101011001100110011111100100101111001111001000101100 010
\
0010011100111000110001001010100011101100000001100111110001101011101000110010 00
\
0011110

129,

0101100111000001111100001001100000001100011101111110110000000000010000001110 01
\
1111011010000101011110001101100000100011011101011011001000101100101101110010 11
\
1111100

128,

10110011100000111110000100110000000110001110111111011000000000001000000111001 1
\
11101101000010101111000110110000010001101110101101100100010110010110111001011 1
\
1111000

127,

01100111000001111100001001100000001100011101111110110000000000010000001110011 1
\
11011010000101011110001101100000100011011101011011001000101100101101110010111 0
\
0111001

126,

11001110000011111000010011000000011000111011111101100000000000100000011100111 1
\
10110100001010111100011011000001000110111010110110010001011001011011100101110 0
\
1110010

125,

10011100000111110000100110000000110001110111111011000000000001000000111001111 1
\
01101000010101111000110110000010001101110101101100100010110010110111001011100 0
\
0101101

124,

00111000001111100001001100000000110001110111111011000000000001000000111001111 10
\
11010000101011110001101100000100011011101011011001000101100101101110010111000 1
\
0010011

123,

01011010010010101111110000110111110111010111010111000011001011001111001000000 0
\
00011001101010101100011110000001001001100001010111000001110101100011101011100 1
\
1100110

122,

1001111010100011001000100101101101111010011001010100010101100101001011111111101
\
1000101110100001011111101000101110110111010100101100100101010111100001001010000
\
0001100

121,

0011110101000110010001001011011011110100110010101000101011001010010111111111011
\
0001011101000010111111010001011101101110101001011001001010101111000010010100001
\
1010001

120,

0101000010111010010100110101100100101001000110111101011010101000011101000010111
\
1001011001110001000010111010011100100110001100100110111111010010111100011111000
\
1100010

119,

1010000101110100101001101011001001010010001101111010110101010000111010000101111
\
0010110011100010000101110100111001001100011001001101111101001011110001111100011
\
1000100

118,

0110100011011111100101110101000001100100111000011001100110011101000110110100111
\
1110000100110000110111110001010101100011101100001111010001101100011111101110011
\
0000001

117,

1111101110001001111101001001010000001001010011011111000000000110111111010110101
\
0111101010010101010011111010001100111100000110001010001000100011000011001010011
\

1000010

116,

1111011100010011111010010010100000010010100110111110000000001101111110101101000
\
1111010100101010100111110100011001111000001100010100010001000110000110010100100
\
1001101

115,

1110111000100111110100100101000000100101001101111100000000011011111101011010010
\
1110101001010101001111101000110011110000011000101000100010001100001100101001000
\
1010011

114,

1101110001001111101001001010000001001010011011111000000000110111111010110100110
\
1101010010101010011111010001100111100000110001010001000100011000011001010010000
\
1101111

113,

1011100010011111010010010100000010010100110111110000000001101111110101101001110
\
1010100101010100111110100011001111000001100010100010001000110000110010100100000
\
0010111

112,

0101101100001000010010001011010111101001001100001100001111100011011001101100100
\
1110101001011101000001011110111001111000011011010000011101001101001100101111010
\
0100111

111,

1001110000100110010010110101111100010010111011110100010011111010000001100110000
\

01101100010011101111101001010101000010111010001101010111110011110011101010111
\
0001110

110,

00111000010011001001011010111110001001011101111010001001111101000000110011000
\
11011000100111011111010010101010000101110100011010101111100111100111010101111
\
1010101

109,

01110000100110010010110101111100010010111011110100010011111010000001100110000
\
10110001001110111110100101010100001011101000110101011111001111001110101011111
\
0101010

108,

11100001001100100101101011111000100101110111101000100111110100000011001100001
1
\
01100010011101111101001010101000010111010001101010111110011110011101010111111
0
\
1010100

107,

11000010011001001011010111110001001011101111010001001111101000000110011000011
0
\
11000100111011111010010101010000101110100011010101111100111100111010101111110
0
\
1100001

106,

10101110111111111101100011101011010011101011001100101110001111100000001111100
00
\
00110001001010111011101100101000100011110001001110110011000111001010011010001
1
\
1001011

105,

01110111110010011011100110011001111110100100001001111011110001001100010001110
\
11011010101000111000011111011000111001010101111000101100110000101011110001110
\
0011111

104,

11101111100100110111001100110011111101001000010011110111100010011000100011101
1
\
10110101010001110000111110110001110010101011110001011001100001010111100011101
0
\
0111110

103,

11110101000100000011110001010011001010011000011100101100001011111101101000101
0
\
11010010011110101110111011101010011011100000000111111001111100010000000010111
0
\
1110101

102,

11000000000101101010001010010010100100111000000010011011011000110111111110100
0
\
00011100000000010010110001011101001001110111101010111001000110011111000000011
1
\
1100011

101,

10000000001011010100010100100101001001110000000100110110110001101111111101000
0
\
00111000000000100101100010111010010011101111010101110010001100111110000000111
0
\
0001111

100,

00101010011011000101000001111110100011101000110010101110101100010011010101110
1
\
11001000111100000100000011111101011001101001001110101110100111000011000100011
0
\
0010111

99,

01111110111011100111101011001001110111011001011110011111001011110101000010001101
\
00101001000101000111000001110011001101100101111000010111110000111001001101011
\
1101110

98,

11111101110111001111010110010011101110110010111100111100101111010100001000110
\
01010010000101000111000001110011001101100101111000010111110000111001001101011
\
1011100

97,

11010001100011111001100010001001110110110110100001011101001000110010011111100101
\
00011100101001010011000001000101001000100000000100010101111110101101111000001
\
0110001

96,

10100011000111100110001000100111011011011010000101110100100011001001111100101
\
00111001010010100110000010001010010001000000001000101011111010110111100000101
\
0101011

95,

01101100000001010000111100111101000011011110011000010101000100101111101011010
\
11001010011000000011000010011101011100110111110100011101001011010000000100110
\
1011111

94,

11110010001000101110011011000000111101110001011010010111011101110010000010111
\
00101100001101001001000010110011000111011000001101110000101000011111001100001
\

1111110

93,

1100111001110011000101111011010100101110101000111110110111010010100010101000 01
\
1110000010011101110100001110111111000000011111111010101110111000000101110111 11
\
1110101

92,

1011011011010000111101010101111010011101110010010001100010011001110111101111 10
\
0111100111001111010100000101011001111011100001100001110110001011110111111001 01
\
1100011

91,

0100011110010111001100001000100111111011000111001111001000001111011101100000 01
\
0100101101101010010100010010010100001100011101010111000111101100010011100100 01
\
1001111

90,

1010010100011000101110110010011100110110101101110010011100100010001001111111 11
\
0010111000100000010100111100001111100011100100111010100100100011011011011110 00
\
1011110

89,

0100101000110001011101100100111001101101011011100100111001000100010011111111 10
\
0101110001000000101001111000011111000111001001110101001001000110110110111100 00
\
1110101

88,

1011111001010100001101101010100000011010010100100101111110110100010101000000 01
\

00000000011101011011111010000110011101010011011111101110011101100100011011101
0
\
0101010

87,

01111100101010000110110101010000000110100101001001011111101101000101010000000010
\
00000000111010110111110100001100111010100110111111011100111011001000110111010
1
\
0011101

86,

11010011011001100000000001001010010101001110001111011110111011101100110111111001
\
10111001001000100000101110010000001011111010011011110011001000101110101011000
1
\
1111010

85,

10100110110011000000000100101001010100111000111101111011110111011001101111110011
\
01110010010001000001011100100000010111110100110111100110010001011101010110001
0
\
0111101

84,

01001101100110000000001001010010101001110001111011110111011101100110111111001
10
\
11100100100010000010111001000000101111101001101111001100100010111010101100010
1
\
0110011

83,

10011011001100000000010010100101010011100011110111011101110110110011011111001
101
\
11001001000100000101110010000001011111010011011110011001000101110101011000101
0
\
1100110

82,

000111000101011011010011011111100101110011110101000111011110010101110101100110
\
001010101101010001001000100010110000000100010110011111000110101010101110100111 1
\
1000101

81,

000100101001101101111100110010000111100101100100111110001111011000100000110001
\
111011010101110001100000100111111111100101010101101110110101000101001011000100
\
1001010

80,

001001010011011011111001100100001111001011001001111100011110110001000001100011
\
110110101011100011000001001111111111001010101011011101101010001010010110001001
\
0010100

79,

010010100110110111110011001000011110010110010011111000111101100010000011000111
\
101101010111000110000010011111111110010101010110111011010100010100101100010010
\
0101000

78,

101111101110110100111100011101110000101110101001000001001000110111001101110010
\
110100100001011111110101011101100011000111010100100100000111000110101001111111
\
0010000

77,

011111011101101001111000111011100001011101010010000010010001101110011011100101
\
101001000010111111101010111011000110001110101001001000001110001101010011111111
\
1101001

76,

11010001100000100010101111101000111011100010101011010001000010111111110011011 0
\
11110000101010110010010001010001001111000010101100000101100111101010101101001 00
\
0010010

75,

10001001001100101010001101111001010001110011011011011000010010101100110010010000
\
01011001101000101011100100101011100000110010111101011100100000010101110001001 0
\
0101101

74,

00010010011001010001101111001010001110011011011011000010010101100110010010000 0
\
10110011010001010111001001010111000001100101111010111001000000101011100010010 1
\
0010011

73,

00001110111111001110110110101000001011001111100011010001111001000000000010111100
\
11011110011111100001010100100111111101111100010000111000111111101000000001000 1
\
1100110

72,

00110111110011110000000101110101101001110100100001001100000111001100111000010 0
\
00000100000010001101101111000110000101001111000100111011000001101111000111100 0
\
0001100

71,

01101111100111100000000101110101101001110100100001001100000111001100111000010 00
\
00001000000100011011011110001100001010011110001001110110000011011110001111000 0
\

0011000

70,

1111010100001010110111111110001001011101101011111111001101001111111100111011010
\
1010100011010111100111101001000110101000101111011010011011100000001101101110110
\
1110000

69,

1110101000010101101111111100010010111011010111111110011010011111111001110110110
\
0101000110101111001111010010001101010001011110110100110111000000011011011101100
\
0101001

68,

1101010000101011011111111000100101110110101111111100110100111111110011101101100
\
1010001101011110011110100100011010100010111101101001101110000000110110111011010
\
0011011

67,

1000001001100000000100101001001100010111011111000101011001010000110101011001000
0
\
1111111001001000000001010000010010111110100101000111110111111010010001100000000
\
0111111

66,

0000010011000000010010100100110001011011111000101011001010000110101011001000010
\
1111110010010000000010100000100101111101001010001111101111110100100011000000010
\
0110111

65,

0010001110110110010011101010110001110111010010111010011000110001100100101111100
\

01000001110101001110010110011011000000010010100010111101000100101110100101100
1
\
0101110

64,

01101101010110100100011101101100001011100001100110001111010111111110111000000
1
\
00111011010111010011101010111111111110010010100000110000110111100010001110100
1
\
0011100

63,

11110000100000100101010011101100100111001011111011101110110000011000101111111
11
\
11001110010011101000010011111011000001001001010010010101101000111101101100010
01
\
1111000

62,

11001011001100100111001111101101111110011111010101111000001110101110010000001
0
\
00100100011010011111100001100101111010010010101100011100011101001001110100100
1
\
1111001

61,

10010110011001001110011111011011111100111110101011110000011101011100100000010
0
\
01001000110100111111000011001011110100100101011000111000111010010011101001001
0
\
0111011

60,

00101100110010011100111110110111111001111101010111100000111010111001000000100
0
\
10010001101001111110000110010111101001001010110001110001110100100111010010010
1
\
0111111

59,

01110011101001010100010101011011000011110010010100000010111010111110101110110
0
\
10011011101110110011001010100110101100100010000110101001010111110001100001000
1
\
0111110

58,

11001101011111000101000010000010110111101100010011000110111010110001110010010
0
\
10001111100000101001010011000100100111110011101000011000010001011100000111100
1
\
0111100

57,

10110000110011100111101100110001011111010000011101001110111010101111001011010
0
\
10100111111100011101100000000000110001010000110101111010011100000111001010100
0
\
1110001

56,

01001011101010100010110001010110001110101000000001011110111010010010111001010
0
\
11110111000101110100000110001000011100010110001110111110000110110001010000101
1
\
1101011

55,

10111101011000101000001010011000101101011000111001111111011101110100101110101
00
\
01010110110110100111001010011001000110011011111000110110110011011101100100110
0
\
0010110

54,

01010000111100111101111100000101101010111001001000111111011100001111001010101
01
\
00010101010000000001010010111011110010000000010100100111011000000100001100001
0
\
0100101

53,

1000101111010001011001000011111110010111101010101011111011111111000000010101111
\
1001001001110100110110001111111001101011011100110000010000111011011101110111111
\
0001010

52,

0011110110010100000100100100101111101111110110111011111011000010110010010100101
\
1001110000011101010000000111010100101101100111110100001010001101000111111001001
\
0011101

51,

0111101100101000001001001001011111011111101101110111110110000101100100101001010
\
0011100000111010100000001110101001011011001111101000010100011010001111110010001
\
0111010

50,

1101110001100110100100110001101101111111111100000001110000011011111101110110111
\
1100100010000001111100000101110101001101000001000100000011001111100011110010111
\
0110100

49,

1001001011111011111111000000001000111111010011101011001101010011000101100100101
\
0010100111110111000100010011001101100001011100011100101101100100111011110011001
\
1100001

48,

0000111111000000100100010001100001011111000010011101001011001101011100111011001
\
1110101100011010110100111110111100111001100110101101110000110010001011110000011
\

1001011

47,

00110101101101001001111001010101101111001010100110001000000010010000010100111 0
\
01101110110000010101011001010111100010000100110011110010100111111010111101110 0
\
1010110

46,

01000001010111111110011010011111101110011101110111010011001011101100000110000 1
\
01100101011101100101110100100110111010111110000010101111110001001010111110001 0
\
1101100

45,

10000010101111111100110100111111011100111011101110100110010111011000001100001 0
\
11001010111011001011101001001101110101111100000101011111100010010101111100010 1
\
1011000

44,

00000101011111111001101001111111011100111011101110100110010111011000001100001 01
\
10010101110110010111010010011011101011111000001010111111000100101011111100010 10
\
1111001

43,

00001010111111111001101001111111011100111011101110100110010111011000001100001 011
\
00101011101100101110100100110111010111110000010101111110001001010111110001010 1
\
1110010

42,

00111111110010001011001111001111010111010101001111110001110100001101001110101 1
\

11101111100100010010001111100111010001010111001110110110101100010000100111000
\
0100100

41,

01111111100100010110011111001111010111010101001111110001110100001101001110101111
\
11011111001000100100011111001110100010101110011101101101011000100001001110000
\
1001000

40,

11010101000101000001010100001001101101011100000100000100011111111000010101001 0
\
00000110101100000111111000010100111011101011011110010000001111111101011001101 0
\
1010000

39,

10000000000111101111000000100111101010110000110011001011110000111100000101100 1
\
10110101100101000000110110100000001001100001011001101010100001000101110110111 1
\
0101001

38,

00101010000010110011101001111011100101101001011101010100101110110100100100111 0
\
11010011110111001110101011001001101101110101010110011111111100110100101000010 0
\
1011011

37,

01010100000101100111010011110111001011010010111010101001011101101001001001110 1
\
10100111101110011101010110010011011011101010101100111111111001101001010000100 1
\
0110110

36,

10101000001011001110100111101110010110100101110101010010111011010010010011101
\
01001111011100111010101100100110110111010101011001111111111001101001010000100 10
\
1101100

35,

01010000010110011101001111011100101101001011101010100101110110100100100111011 0
\
10011110111001110101011001001101101110101010110011111111110011010010100001001 00
\
0010001

34,

10100000101100111010011110111001011010010111010101001011101101001001001110110 1
\
00111101110011101010110010011011011101010101100111111111100110100101000010010 00
\
0100010

33,

01101011010100011001010101000110000100100110010001010100010101011110110010011 1
\
11000011011010011010100010111111000100011100101010110100100100101011001100101 0
\
1001101

32,

11111100100101011111000010111000111001000100011001101011100101110001001011001 0
\
00111110001001111010000011110111110110000111011000010001111011110100101110011 10
\
1011010

31,

11010011000111010011101101000101000010000000000100001010000010010111011100110 01
\
11000100101110111011000001100110010010101010000100001101010001101101111100011 1
\
0111101

30,

1000110000001100101011001011111011010000100010101110101100011001000101110011101
\
0011000110000011100100010100010101101110001110110101000000111011001001111010100
\
1110011

29,

0011001000101111100000110100100101100000110011011000101010000111011100101100001
\
1101101111110011110100110000001100100111000011111110101000010111011011111110011
\
1101111

28,

0100111001101001110111001010011000000011101110001110100100100001000000000111110
\
0000111100010011010101111000111110110101011001101001111011010101001011101111100
\
0011110

27,

1011011011100101011000110111100011000111111111110001000101111111011001010000001
\
1010011011010010010111101001011010010001101101000111011101010001101011001000011
\
1111100

26,

0100011111111110000011100110001010100111101110000111000011100000101011111111110
\
1111010101010000010011001010010011011000000100011010010001011000101010000111011
\
1110001

25,

1000111111111100000111001100010101001111011100001110000111000001010111111111101
\
1110101010100000100110010100100110110000001000110100100010110001010100001110111
\

1100010

24,

000111111111000001110011000101010011110111000011100001110000010101111111111011
\
110101010100000100110010100100110110000001000110100100010110001010100001110110
\
0001101

23,

000101011101011000111100000111101011101100001001110011010011011000110100001010
\
000100100111011010010100101011110011101111110100011010000011111010110010110111
\
1011010

22,

001010111010110001111000001111010111011000010011100110100110110001101000010100
\
001001001110110100101001010111100111011111101000110100000111110101100101101111
\
0110100

21,

011111010110111000101010010011100010110010101001111101111100100000110110101010
\
111100010010111010100011001101010001010010101000111010100000000100111010000101
\
0101000

20,

111110101101110001010100100111000101100101010011111011111100100000110110101011
\
111000100101110101000110011010100010100101010001110101000000001001110100001010
\
1010000

19,

110111111000111001110011000011000111001000101001000111001010110010100110101010
\

01111100010011100111110101011101101010011101101011100010111111110001100100111
\
0101001

18,

10111111000111001110011000011000111001000101001000111001010110010100110101010
\
11111000100111001111101010111011010100111011010111000101111111100011001001111
\
0011011

17,

01111110001110011100110000110001110010001010010001110010101100101001101010100
\
11110001001110011111010101110110101001110110101110001011111111000110010011111
\
1111111

16,

11010110010001010100001001010111010100011100011000100110010110011111111010111
0
\
01011010100001110001101101100100101101011010111001011101000000110011100010010
0
\
0111110

15,

10101100100010101000010010101110101000111000110001001100101100111111110101110
0
\
10110101000011100011011011001001011010110101110010111010000001100111000100100
1
\
0110101

14,

01011001000101010000100101011101010001110001100010011001011001111111101011100
1
\
01101010000111000110110110010010110101101011100101110100000011001110001001001
1
\
0100011

13,

101100100010101000010010101110101000111000110001001100101100111111110101110010
\
110101000011100011011011001001011010110101110010111010000001100111000100100110
\
1000110

12,

010011100110001011111111010000011101110011101100101001101010001100100000011000
\
000100001000010101000111110000101010000110011100100110101100100001111000010111
\
0000101

11,

101101101111001100100100101101110111100101010111100011100111101010001011001101
\
100110011111111001111110000011001011100001000000011111110110101100000001110101
\
1001010

10,

010001111101000010010011010110100011001000100001110111111100100111011101100110
\
100010110000100000001101100100001000101111111001101101000010110111111001011000 1
\
0011101

9,

100011111010000100100110101101000110010001000011101111111001001110111011001101
\
000101100001000000011011001000010001011111110011011010000101101111100101100010
\
0111010

8,

000111110100001001001101011010001100100010000111011111100100111011101100110 10
\
001011000010000000110110010000100010111111100110110100001011011111001011000101
\
0111101

7,

00111110100001001001101011010001100100010000111011111110010011101110110011010
\
01011000010000000110110010000100010111111100110110100000101101111100101100010
\
1111010

6,

01111101000010010011010110100011001000100001110111111100100111011101100110100
\
10110000100000001101100100001000101111111001101101000010110111110010110001010
\
1110100

5,

11010000001001001011000101110010100001001011010100111010000001110111100010110
\
11011001111101010100001110011000100001000100111111001111010001011010100111000
\
0101000

4,

10100000010010010110001011100101000010010110101001110100000011101111000101100
\
10110011111010101000011100110001000010001001111110011110100010110101001110000
\
0011001

3,

01101010101001000001111111111111011010010010110100010101100100000100101001001110
\
11011111001000011111111111110101111101010010001100111011111101101010101100110000
\
0111011

2,

11111111011111101110010111001001011001000011101010010101011111101001100110000
\
00000110101101110000111001011110001011111111010110100101001000010101110110101 1
\

0110110

1,

1101010011001011000100011010011000001000111110111110100111000001111110001111011
\
1011010110011010111011010011010110100100100100100000000001011100101001010001100
\
1100101

0,

1010100110010110001000110100110000010001111101111101001110000011111100011110111
\
0110101100110101110110100110101101001001001001000000000010111001010010100011000
\
0000011

As can be seen in the last iteration of the loop the result of this complete reduction multiplication is:

polynomialToBinary(ab, f);

1010100110010110001000110100110000010001111101111101001110000011111100011110111
\
0110101100110101110110100110101101001001001001000000000010111001010010100011000
\
0000011

At this point, to mirror the calculation which we performed

without complete reduction, we can add the completely reduced value c:

abc := addPolynomial(ab, cr):

Here we see that the final mathematical value of the two methods are the same, modulo the irreducible, although the wordsize method separated the full reduction out so that is need only be applied when desired (for example to supply the unique representation used in an ECC algorithm). Many other operations (such as additions, squarings and inversions) might also be performed without full reduction; only the final result need be reduced to comply with standard elliptic curve methods such as signatures or key agreements. The final reduced values are the same. After final reduction there is zero difference between the results:

(abcwr - abc) mod 2;
0

## APPENDIX C – RELATED PROCEEDINGS APPENDIX

**NONE**

21599720.1